基于织女星开发板的智能门锁控制

1. 基本介绍

该文档描述了一种基于织女星开发板（RV32-VEGA-Lite）的智能门锁例程设计。其采用蓝牙 v4.2 版本的协议栈，运行于 RV32M1 微处理器上的 ZERO_RISCY 核上。该例程提供的内容包括：1）预先编译可直接运行于 RV32M1 开发板上的二进制 bin 文件；2）可运行的 C 工程文件。

该文档首先描述了例程在织女星开发板上的使用环境，包括软硬件条件，其次介绍了低功耗蓝牙技术以及整个系统的设计框图。随后对于例程的代码部分做了详细介绍。文档最后，介绍了例程与手机端应用程序之间通信的详细操作方式。

目录

2 概述

该例程设计了一个智能门锁应用，在该应用中，门锁可以通过蓝牙与手机应用端进行通信，从而使手机可以控制门锁的开关，控制方式有自动和手动两种模式。该例程包括一个织女星开发板，一个由 H 桥驱动的直流电机以及一部支持蓝牙的智能手机。

该例程采用蓝牙 v4.2 版本的协议栈，该协议栈作为中间件包括在 RV32M1 的 SDK 中。例程代码包括协议栈代码和应用代码两部分。当设备工作时，RV32M1 微处理器收到并解析来自手机发送的命令，随后根据命令控制两个 GPIO 口从而达到控制开关的作用。在手动模式，用户通过点按手机应用上的“LOCKED”和“UNLOCKED”按钮来完成对门锁的控制。在自动模式，用户通过将手机拿远拿近以改变蓝牙信号的接收强度从而控制门锁的开关。

在上电之后例程会以 20ms 到 30ms 的间隔进行广播，但出于低功耗的考虑，在 30s 之后广播间隔会更改为以 1s 为间隔。

3 开发环境搭建

该节介绍了开发过程中所需的软件开发环境以及例程实际使用中所需的硬件环境。

3.1 软件开发环境搭建

根据 www.open-isa.cn 上的文档《织女星开发板嵌入式开发环境搭建》，在 windows 下安装以下软件工具：

- Eclipse 集成开发工具
- GNU MCU Eclipse Windows Build Tools (可选的)
- RV32M1 GNU GCC 工具链
- OpenOCD 调试软件
- RV32M1 软件开发包SDK

3.2 硬件开发环境搭建

准备以下设备：

- 织女星开发板, 可以从 www.open-isa.cn 网站订购获得
- H桥驱动的直流电机
- 支持低功耗蓝牙的智能手机且安装好门锁应用

4 整体框架介绍

该节介绍了低功耗蓝牙技术以及整个智能门锁例程的系统框架。

4.1 低功耗蓝牙介绍

该例程使用的低功耗蓝牙协议栈基于蓝牙 v4.2 版本。低功耗蓝牙是对经典蓝牙的一种补充，它主要用于实现无线传输的低功耗化。低功耗蓝牙的协议栈分为三部分：控制层（controller），主机层（host），应用层（application）。控制层是设备的物理部分，负责接收和

发送无线信号完成射频信号和数字信号之间的转换。主机层是软件协议部分，负责设备之间的通信建立与维护。应用层建立在控制层以及主机层之上，面向使用者，由用户定义应用。

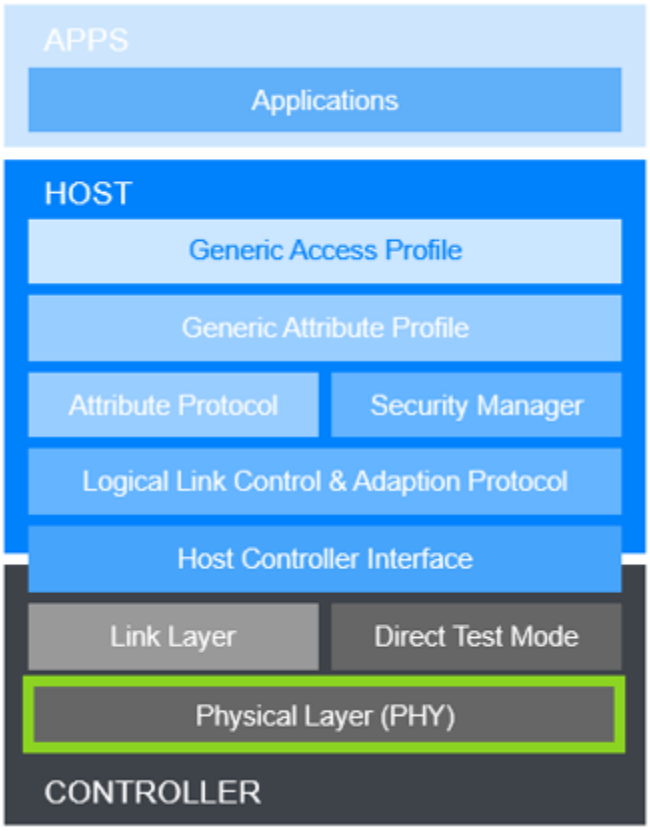

图 4. 1 低功耗蓝牙协议栈

控制层包括物理层，链路层，直接测试模式以及主机控制接口层的下半部分。主机层包括逻辑链路层，适配协议，属性协议以及安全管理协议，通用属性规范以及通用接入规范。低功耗蓝牙的工作射频频段为 2.4GHz。在 RV32M1 微处理器上，使用高斯频移键控（GFSK）作为调制解调方式，其码率为 1m/s。

4.2 系统框图

例程的系统框图如下图所示：

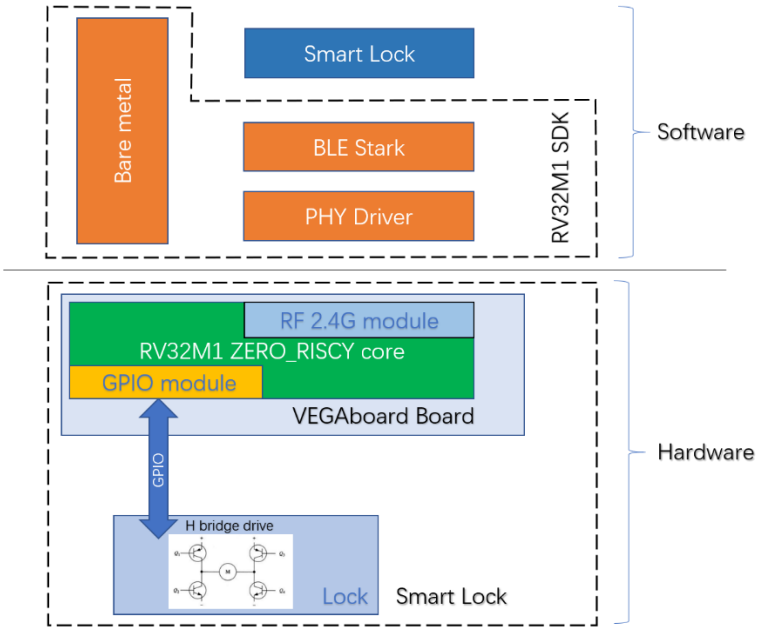

图 4.2 系统整体框架

整个例程基于蓝牙 v4.2，它以中断结合 bare metal 的方式对多任务进行处理。门锁以直流电机驱动，电机由一个 H 桥电路驱动。RV32M1 微处理器通过两个 GPIO 口的输出控制 H 桥从而达到控制电机的目的。整个例程工作在 2.4GHz 频段下。

5 代码架构介绍

该节描述了代码的工作流程以及代码的组织形式，对于例程的自定义部分也做出了介绍。

5.1 代码入口

例程的起点位于 *main* 函数, 该函数位于 *framework/OSAbstraction* 下的 *fsl_os_abstraction_bm.c* 中。

```
int main (void)
{
    extern void BOARD_InitHardware(void);

    OSA_Init();
    /* Initialize MCU clock */
    BOARD_InitHardware();
    OSA_TimeInit();
    OSA_TaskCreate(OSA_TASK(main_task),NULL);
    OSA_Start();

    return 0;
}
```

main 函数首先初始化一些系统资源，然后调用 *OSA_TaskCreate* 函数建立一个 *main_task* 。在 *main_task* 最后，调用 *App_Thread* 函数。

```
void main_task(uint32_t param)
{
  ……
  ……
   /* Call application task */
    App_Thread( param );
}
```

App_Thread 函数主要用来处理该应用中产生的各种 *event* 事件，包括定时事件，信号量事件以及其它用户定义的事件。在该函数中，调用 *OSA_EventWait* 函数以等待事件发生，一旦事件发生 *App_HandleHostMessageInput* 函数就会被调用以做相应的处理。

```
void App_Thread (uint32_t param)
{
  ……
  OSA_EventWait(mAppEvent, osaEventFlagsAll_c, FALSE, osaWaitForever_c , &event);
  ……
   /* Process it */
  App_HandleHostMessageInput(pMsgIn);
  ……
  }
```

App_HandleHostMessageInput 函数是一个switch-case 结构, 和本例程相关的分支为 *gAppGattServerMsg_c*.

```
static void App_HandleHostMessageInput(appMsgFromHost_t* pMsg)
{
  ……
  case gAppGattServerMsg_c:
        {
          if (pfGattServerCallback)
            ……

         }
  ……
  }
```

函数指针 *pfGattServerCallback* 在 *BleApp_Config* 函数中被注册指向 *BleApp_GattServerCallback* 函数.

```
static void BleApp_Config()
{
  ……
  App_RegisterGattServerCallback(BleApp_GattServerCallback);
  ……
  }
```

BleApp_GattServerCallback 函数根据接收到的命令调用门锁驱动实现开关操作。

```
static void BleApp_GattServerCallback (deviceId_t deviceId, gattServerEvent_t*
pServerEvent)
{
  ……
  if (handle == value_smart_lock_ctrl_point)
   {
   if (SMART_LOCK_STATUS_LOCK == val)
     smart_lock_lock(450);
   else
     smart_lock_unlock(450);
  ……
   }
  ……
}
```

main 函数中, 在初始化及创建任务之后需要调用*OSA_Start* 函数，该函数是一个用于多任务之间切换的for循环。

```
void OSA_Start(void)
{
  ……
   for(;;)
   {
  ……
   }
  ……
}
```

整体代码的流程图如下:

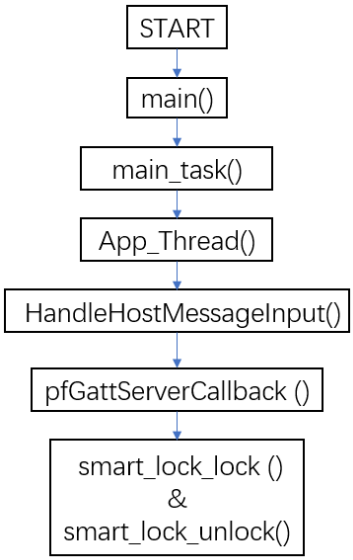

图5.1 代码流程图

5.2 代码的结构

代码可以分为三个部分，一部分在 *sources* 文件夹, 第二部分在 *board* 文件夹，第三部分在 *bluetooth/profiles/smart_lock* 文件夹:

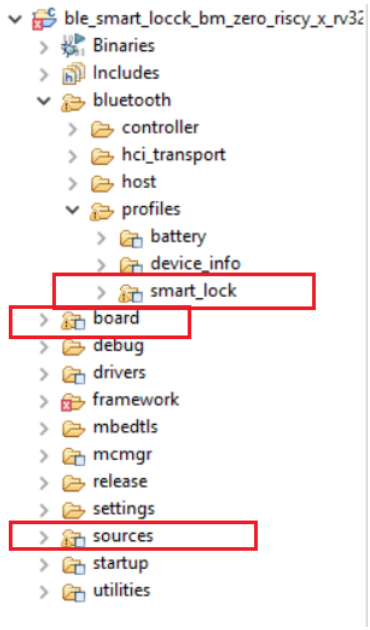

图 5.2 代码结构

smart_lock:

- *smart_lock_service.c* 定义了一些与门锁服务相关的函数。
- *smart_lock_interface.h* 提供了 *smart_lock_service.c* 中定义函数的接口。

sources:

- *app.c* 定义了一些蓝牙协议栈需要使用到的函数，例如 *BleApp_Init(), BleApp_GenericCallback(), BleApp_GattServerCallback(),* 等。
- *app_config.c* 定义了一些和协议栈相关的数据与信息，例如广播数据，默认的广播参数等。

```
app_config.c: change the advertise name
   ……
    {
     .adType = gAdShortenedLocalName_c,
     .length = 8,
     .aData = (uint8_t*)"SML_SML"

    }
   ……
```

- *smart_lock_ctrl.c* 定义了门锁驱动函数以及驱动门锁的 GPIO 口。

```
smart_lock_ctrl.c: change the GPIOs

/* Declare smart lock control GPIO pins */
gpioOutputPinConfig_t smart_lock_ctrl_pins[2] = {
   ……
 .gpioPort = gpioPort_C_c,
 .gpioPin = 28,

   ……
 .gpioPort = gpioPort_C_c,
 .gpioPin = 29,
   ……
   }
smart_lock_ctrl.c: lock driver
   ……
int32_t smart_lock_lock(uint32_t delay_ms){……}
int32_t smart_lock_unlock(uint32_t delay_ms){……}
   ……
```

- *gatt_db.h* 定义了与蓝牙协议相关的服务与特征值，包括 *smart lock service*，该服务包括两个特征值，一个可写用于控制门锁的状态，一个可读用于读取当前门锁的状态。

```
gatt_db.h: define the smart lock service
   ……
  PRIMARY_SERVICE_UUID128(service_smart_lock, uuid_service_smart_lock)
   ……
```

- *gatt_uuid128.h* 定义了 *smart lock service* 的 UUID。

```
gatt_uuid128.h: define the UUID of smart lock service
   ……
  UUID128(uuid_service_smart_lock,0x00, 0x7a, 0x3c, 0x9a, 0x05, 0x00, 0x96, 0x8a,
   0xe6, 0x11, 0x1c, 0x70, 0x60, 0x6c, 0x84, 0xbc)
   ……
```

board:

- *board.h* 定义了一些板级接口以及板级信息。

```
board.h: change the board name
   ……
    /*! @brief The board name */
   #define MANUFACTURER_NAME          "OPEN-ISA"
   #define BOARD_NAME                 "RV32M1-VEGA"
   ……
```

备注: 这些文件使用了eclipse 工程的虚拟文件夹，它们的实际物理地址可以在相应的工程属性里看到。

6 例程使用

1) 将 ble_smart_locck_bm_zero_riscy_x_rv32m1_vega 工程导入 eclipse IDE，路径如下：

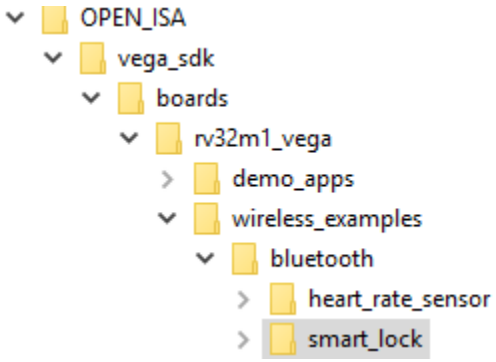

图 6.1 工程目录

2) 将版本设为 release 版本且编译；
3) 根据《织女星开发板快速入门指南》将 RV32M1 芯片设置为从 ZERO_RISCY 启动；
4) 将板子通过 J17 接口与 J-Link 相连，通过 USB 接口 J11 与 PC 相连；

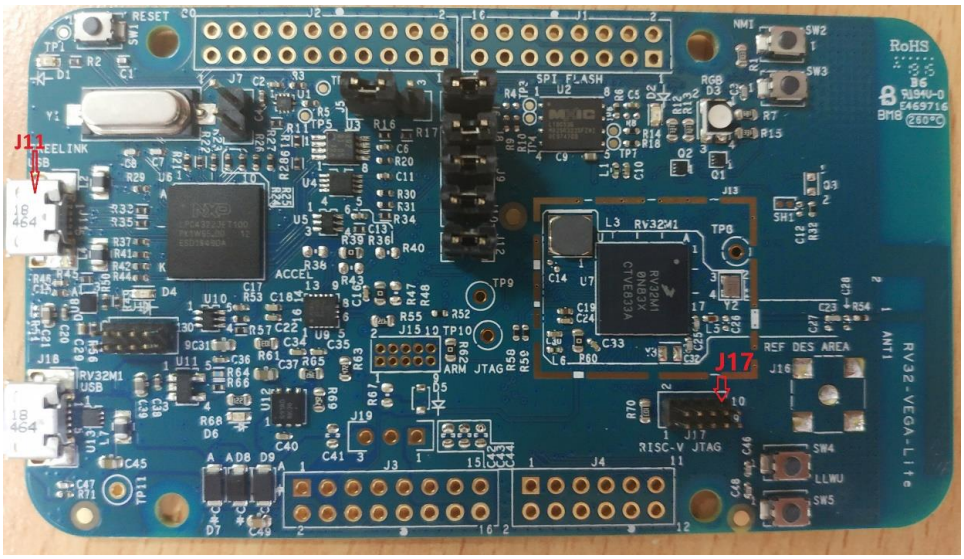

图 6.2 J11 与 J17 接口

5) 调试并下载例程；
6) 下载完成后，移去 J-Link 并断电重新启开发板；
7) 将 PC28（J2-17）和 PC29（J2-15） GPIO 信号与门锁模块相连；

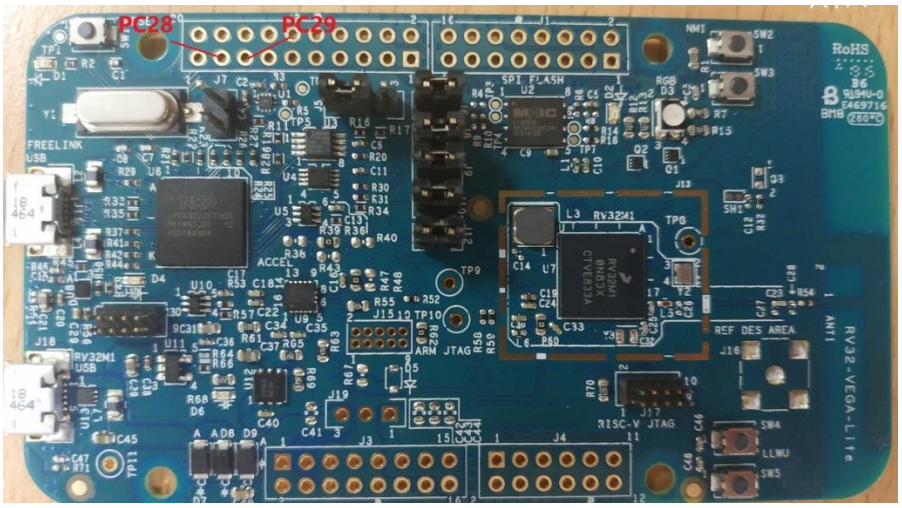

图 6.3 PC28 与 PC29 接口

8) 打开 E-Locker 应用，将看到扫描到的织女星开发板，名字为 “SML_SML”, 点击创建连接之后可以用手动或者自动的模式进行控制；

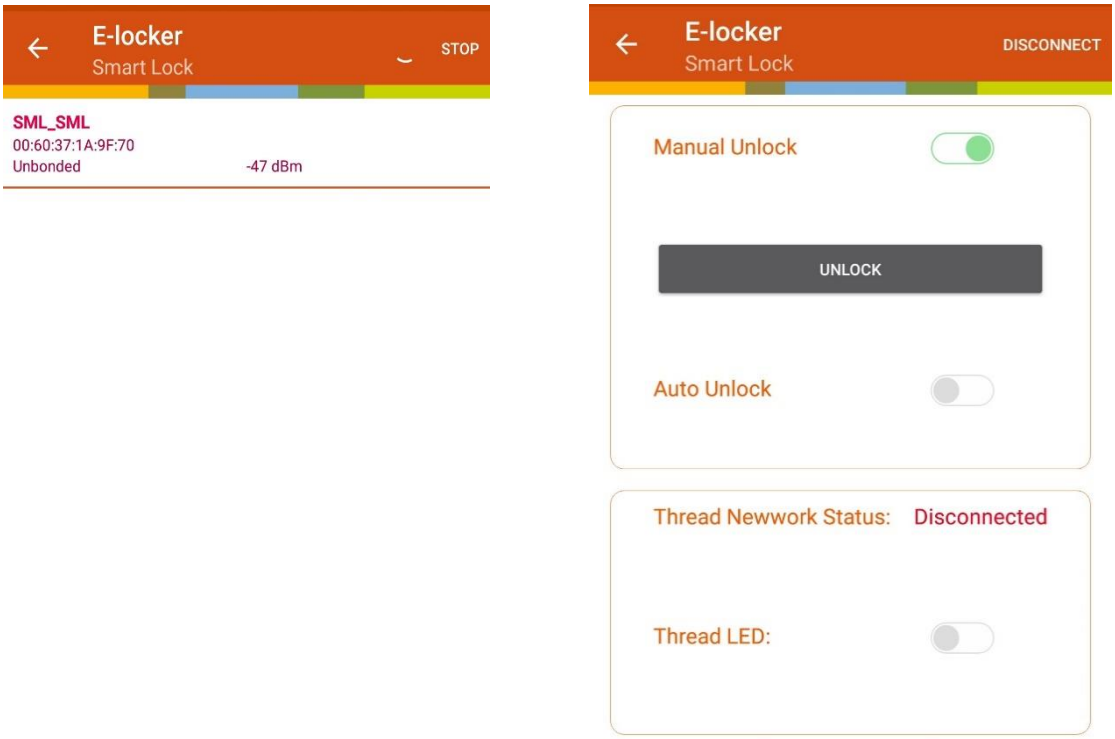

图 6.4 广播名称以及手动控制模式

9) 在自动模式下，开关的状态取决于接受到的 RSSI 强度，当 RSSI 值大时，开关打开，反之则关闭。

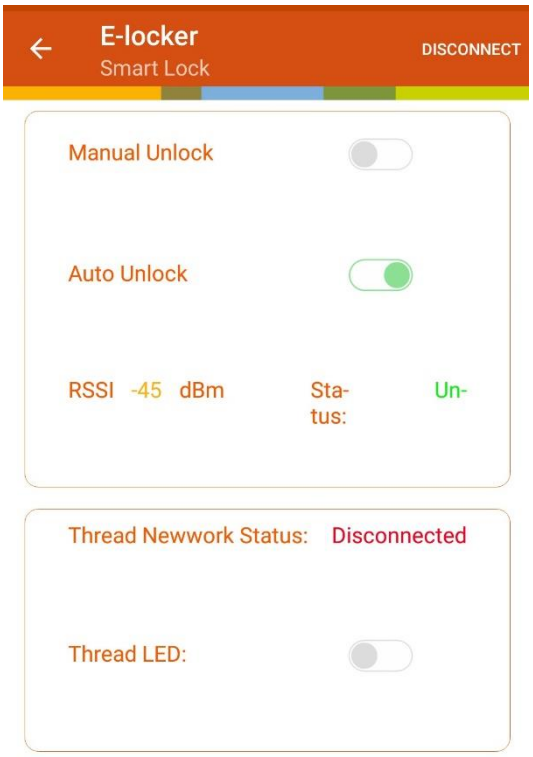

图 6.5 自动控制模式

7 参考文档

- *VEGA-Lite-SCH: 原理图*
- *织女星开发板硬件用户指南.pdf*
- *织女星开发板嵌入式开发环境搭建.pdf*
- *织女星开发板快速入门指南.pdf*

8 发布历史

版本号	日期	修改内容
0	10/2018	初始版本